

ASEN5317 - Computational Fluid Dynamics
Instructor: Professor L. Kantha

Homework #3: Non-Linear Dynamics and Boundary Layer Theory

by

Joseph P. Kubitschek
15 October 1998

ABSTRACT

The primary purpose of this work was to become more familiar with numerical techniques available for solving systems of ordinary differential equations, initial value problems (IVPs), and two-point boundary value problems (BVPs). The first part consisted of investigating non-linear dynamics in the context of the modified Lorenz set of ordinary differential equations. Three solution techniques, available from [Numerical Recipes](#)¹, were used to solve the modified Lorenz set (RKDUMB, RKQS, and BSSTEP). The results duplicate the classic Lorenz trajectories that represent impressive structure in the midst of non-linearity and ultimately led to the development of chaos theory. Improved computational efficiency was obtained with the use of RKQS and BSSTEP, two adaptive step-size algorithms. However, this improvement was at the expense of some loss in structure of the solution trajectories as compared with the use of RKDUMB, a fixed step-size algorithm. The second and third parts consisted of solving the classical velocity and thermal boundary layer similarity equations for flow over a flat plat. The velocity boundary layer similarity equation (i.e. Blausius' equation) was solved as a two-point BVP using RKDUMB and an interval halving scheme. Then, the thermal boundary layer similarity solution was obtained by making use of the preceding solution and the matrix method, TRIDAG. Three values of Prandtl number (Pr) were investigated, $\text{Pr} = 1.0, 0.714 \text{ (air), and } 6.75 \text{ (water)}$.

INTRODUCTION

The purpose of this project is to gain increased familiarity with available numerical solution techniques used to solve systems of ordinary differential equations (ODE), initial value problems (IVP), and two-point boundary value problems (BVP). The Lorenz set of ODEs was solved using RKDUMB, RKQS, and BSSTEP; Blausius' similarity equation for laminar flow over a flat plate was solved using RKDUMB and interval halving; and, the thermal boundary layer similarity equation for laminar flow over a flat plate was solved using TRIDAG.

Background

Part 1: Modified Lorenz System

The modified Lorenz set of ODEs is a non-linear system. Physically, the system represents buoyancy-induced flow in which a fluid is heated at a lower elevation. This heating produces a localized change in density, thereby resulting in a net buoyancy force. The buoyancy force sets the fluid in motion and it rises. The Lorenz set is given as

$$\begin{aligned} \frac{dx}{dt} &= -ax + ay + f\cos(\theta) \\ \frac{dy}{dt} &= -xz + bx - y + f\sin(\theta) \\ \frac{dz}{dt} &= xy - cz \end{aligned}$$

where, x represents the convection intensity; y and z represent the horizontal and vertical temperature variations; and a , b , and c are constants where a is proportional to the Prandtl number (Pr) of the fluid, b is the Rayleigh number (Ra), and c represents the aspect ratio (Kantha- Lecture Notes) for buoyancy-induced flow between two horizontal plates. In general, f represents a perturbation of the Lorenz system. The Lorenz system was obtained by truncation of the governing equations of fluid mechanics and represents a system that is well behaved for low values of Ra, but exhibits chaotic behavior for high values of Ra. Pr, by definition, represents the ratio of momentum diffusion to thermal diffusion (i.e. an indicator of the predominant mode of heat transfer), while Ra represents the ratio of buoyancy forces to viscous forces (i.e. a relative measure of the strength of the buoyancy-induced flow). Thus, for specified values of a , b , c , f , and θ the modified Lorenz system can be solved as an IVP using algorithms available from [Numerical Recipes](#)¹.

Parts 2 and 3: The Velocity and Thermal Boundary Layer Similarity Solutions

The velocity and thermal boundary equations for flow over a flat plate can be solved using boundary layer theory. In this case, simplifying assumptions (i.e. the boundary layer approximations) are applied to full Navier-Stokes and energy equations that govern fluid flow. The result is a set of partial differential equations (i.e. the boundary layer equations) that may be further reduced into a set of two coupled ordinary differential equations. This is achieved by introducing the stream function, ψ , such that continuity is automatically satisfied, and then performing a similarity analysis by defining ψ to be some function of the similarity variable, η . The resulting set of equations is

$$\begin{aligned} F'' + 2F''' &= 0 \\ \theta'' + \frac{1}{2}\text{Pr}F'\theta' + 2\text{Pr}F'' &= 0 \end{aligned}$$

where, $F = f(\eta)$ and $\eta = y(U/vx)^{1/2}$. The necessary boundary conditions are

$$F = F' = \theta' = 0, @ \eta = 0.$$

$F' \rightarrow 1$ and $\theta \rightarrow 0$, as $\eta \rightarrow \infty$.

Although these equations are coupled, the first equation may be solved for F , F' , and F'' as a two-point boundary value problem by making use of the boundary conditions. The solution can then be substituted into the second equation that can subsequently be solved for θ using a suitable matrix method.

Solution Techniques

Part 1: Modified Lorenz System

For the modified Lorenz set of ODEs, the numerical techniques were used (RKDUMB, RKQS, and BSSTEP). The FORTRAN source code for the driver routines and the refined subroutines are included for each part of this study as appendix B. RKDUMB is a refined subroutine that implements RK4, a single Runge-Kutta integration step, with a constant step size. The user must provide the subroutine code *derivs.f* that specifies the set of ODEs to be solved. RK4 then calls this subroutine, implements the Runge-Kutta method, and returns updated values of the solution. This procedure is repeated until the trajectory reaches the upper limit of the interval in question. Alternatively, RKQS and BSSTEP are adaptive step size algorithms that make use of the results from the previous step and optimize the step-size for the following step. The advantage in general is improved efficiency with respect to the number of function evaluations required to obtain the solution (i.e. computational expense) at some *a priori*-specified accuracy. RKQS implements a single fifth-order Runge-Kutta step and calls the subroutine RKCK that advances the solution over the interval in question and returns information on the error. RKQS and BBSTEP were implemented for this project using a driver program that calls ODEINT, a subroutine that makes use of either RKQS or BSSTEP. Finally, BSSTEP makes use Burlisch-Stoer method, an algorithm that implements Richardson extrapolation and improves subsequent step-sizes using the information from the previous step-size error.

Although RKQS and BSSTEP provide improved accuracy and efficiency over RKDUMB, the latter in general is a sound algorithm for solution to problems that involve smooth functions and in which case a high degree of accuracy is not required. This is certainly the case for the boundary layer similarity equations in the context of this study. Thus, RKDUMB was used for part two in solving the velocity boundary layer similarity equation as a two-point BVP.

Parts 2 and 3: The Velocity and Thermal Boundary Layer Similarity Solutions

The boundary layer similarity equations were solved using the refined algorithms RKDUMB and TRIDAG. As previously mentioned RKDUMB implements the fourth-order Runge-Kutta method. Although in general, this algorithm has limited accuracy, the use of interval halving can improve this situation and was implemented in the driver program for RKDUMB in solving the velocity boundary layer similarity equation. The first step was to reduce the similarity equation (a third order ODE) to a system of three, first-order ODEs, the result of which is given as

$$\begin{aligned} dF/d\eta &= p, \\ dp/d\eta &= q, \\ dq/d\eta &= -1/2(Fq). \end{aligned}$$

The necessary boundary conditions are obtained as $F = F' = 0$ at $\eta = 0$, and $F' \rightarrow 1$ as $\eta \rightarrow \infty$. Next, the integration interval ($\eta = 0 - 10$) was divided into equal steps giving a step-size of 0.2. Following each fourth-order Runge-Kutta solution over the interval, interval halving was used to

obtain improved starting values and ultimately improved accuracy to the *a priori* specified $\text{eps}=10\text{e-}6$.

Following solution of the hydrodynamic boundary layer equation, the solution was substituted into the thermal boundary layer equation, which was subsequently solved using the tri-diagonal matrix method as the refined Numerical Recipes¹ algorithm TRIDAG.

RESULTS AND DISCUSSION

Part 1: Modified Lorenz System

The solution trajectories for the modified Lorenz system, solved using RKDUMB, RKQS, and BSSTEP are included as appendix A. For the case when $f = 0$, the results are presented as x, y, and z verses time; x verses y; x verses z; y verses z; and in 3-dimensional x-y-z space. Figures 1-3 represent the time series plots of the solution trajectories in x, y, and z space, respectively. (Note: Although these time series results were initially plotted over the full time domain for which the problem was solved, the structure exhibited by the trajectory is more easily visualized by plotting the initial 150 seconds of the trajectory). In each case, the transitory nature of the trajectories can easily be seen. Although random in general, the oscillations in time occasionally appear ordered with a non-linear growth in amplitude followed by a return to chaotic behavior at some critical amplitude. The chaotic behavior is then observed for some time until it again randomly transitions to a similarly stable state, exhibiting a non-linear growth rate. This process is repeated in random or chaotic fashion. The nature of this chaotic behavior is reflected in the transition between the apparently stable growth rate in amplitude and the disordered random variation in amplitude from step to step, the recognition of which, by Lorenz, led to theory of chaos. Although some structure can be seen in figures 1-3, a better visualization of the structure is obtained by plotting the trajectories in x-y, x-z, and y-z space (i.e. trajectory projections). Figures 4-6 represent the trajectory projections in each of the three variable spaces and provide an interesting look at the nature of the phenomenon. Figure 4 represents the classic Lorenz butterfly and demonstrates the “strange attractor” concept exhibited by the solution trajectory in the midst of the chaotic behavior. The degree of structure here is quite impressive, and provides an indication of the duration of time the trajectory spends in oscillation about some value. Although the transition between the two states is random (as seen in the time series plots), it is apparent that the trajectory spends approximately equal times in each state. This feature is realized in comparing the line densities of the two states. Figures 4 and 5 show similar features as those of figure 4. Alternatively, Figure 7 represents the 3-dimensional plot of the solution trajectory over the time intervals, 0-50 seconds, and provides a 3D perspective of the trajectory structure. Furthermore, the location of the “strange attractors” in 3-dimensional space can be visualized from this presentation of the results.

The solution trajectories were also obtained for the RKQS and BSSTEP algorithms. Figures 8-10 represent the trajectory time series plots that look much the same as those for the case of RKDUMB. Figures 11-13 represent the projections, obtained using RKQS, in x-y, x-z, and y-z space, respectively. While figures 14-16 represent the time series and figures 17-19, the trajectory projections, obtained using BSSTEP, also in x-y, x-z, and y-z space, respectively. In comparison of these results with those obtained using RKDUMB, it is easily seen that although similar features exist, the “smoothness” in structure is lost. This is most certainly a result of using the adaptive step size algorithms, RKQS and BSSTEP that increase efficiency at the loss of details associated with the trajectory. For RKQS the number of function evaluation required was obtained as 24,318. This is quite good in comparison with that for BSSTEP obtained as 71,159 function

evaluations. Thus, even though improved efficiency and accuracy are obtained with adaptive step-size algorithms, they do not characterize the structure of trajectories as well as those obtained using RKDUMB, in this case. However, a smoothing technique, likely available with many plotting packages, could be applied in the cases of RKQS and BSSTEP to improve the results and hence recover some of the lost structure. But, the use of a smoothing technique would increase computational space and hence represents degraded efficiency. Therefore, RKDUMB proves to be a better algorithm in this case.

Next, the Lorenz system was solved using RKDUMB for the perturbed cases of $f = 2.5$ and $\theta = 60^\circ$ and 240° . Figures 20-22 represent the time series plots while figures 23-25 are the trajectory projections for $\theta = 60^\circ$, and figure 26 is the 3D trajectory plot. Figures 27-29 are the time series plots and figures 30-32 are the trajectory projections; while figure 33 represents the 3-D trajectory plot for 240° . Physically these two cases can be thought of as two different climatic states. In general, the nature of introducing the perturbation is seen in comparison with figures 4-6 as the trajectories spend more time in one state or the other depending on the value of θ . For the two cases here, they are exactly 180° out of phase and observation of the line densities reveals that the trajectories do indeed spend more time in opposite states. The fact that they are exactly out of phase is revealed by inspection of the respective line densities of each resulting Lorenz butterfly.

Finally, a constant $g = 2.0$ was added to the third ODE in the Lorenz set and RKDUMB was used again to obtain the solution trajectory as in the preceding results. Figures 34-36 and 37-39 represent the time series and trajectory projections for the case of $\theta = 60^\circ$, with figure 40 being the 3-D trajectory plot; and figures 41-43, 44-46, and 47 are the respective plots for $\theta = 240^\circ$, respectively. As can be seen by these results, the perturbation forces the trajectory to spend even more time in one state or the other depending on θ . Thus, introducing g into the set of equations produces a stabilizing effect.

Part 2: The Velocity Boundary Layer Similarity Solution

The solution to Blasius' equation, as previously mentioned was obtained by solving the two-point BVP using the fourth-order Runge-Kutta method (i.e. RKDUMB). The results provide an indication of the nature of boundary layer growth for classical case of viscous flow along a flat plate and have been included as table 1. Comparison of these results with those obtained by L. Howarth in Boundary Layer Theory⁴ (Schlichting, 1979) show exact agreement to the fifth decimal place. Undoubtedly this is as expected since Howarth used the same fourth-order Runge-Kutta method used here. An important parameter, the boundary layer thickness, δ may be obtained by noting that the similarity solution for F' asymptotically approaches 1.0. Then, by defining the boundary layer thickness to be adequately represented when $u=0.99U_\infty$ (i.e. $F' = 0.99$), the value of η can be obtained from table 1 as approximately 5.0. And, since $\eta = y(U_\infty/vx)^{1/2}$ as defined in the similarity analysis, the boundary layer thickness in this case is obtained by solving for y as $\delta = 5.0(vx/U_\infty)^{1/2}$. But the Reynolds number (Re) is defined as $Re_x = U_\infty x/v$, hence the boundary layer thickness may be written in non-dimensional form as $\delta/x = 5.0/Re_x^{1/2}$, the classical result from boundary layer theory.

Table 1. – Hydrodynamic boundary layer similarity solution.

η	F	F'	F''
.0000	.000000	.000000	.332057
.2000	.006641	.066408	.331984
.4000	.026560	.132764	.331470

.6000	.059735	.198937	.330079
.8000	.106109	.264709	.327389
1.0000	.165572	.329780	.323007
1.2000	.237949	.393776	.316589
1.4000	.322982	.456261	.307865
1.6000	.420322	.516756	.296663
1.8000	.529519	.574758	.282931
2.0000	.650025	.629765	.266752
2.2000	.781194	.681310	.248351
2.4000	.922291	.728981	.228092
2.6000	1.072507	.772454	.206455
2.8000	1.230978	.811509	.184007
3.0000	1.396809	.846044	.161361
3.2000	1.569095	.876080	.139128
3.4000	1.746951	.901760	.117877
3.6000	1.929526	.923329	.098087
3.8000	2.116030	.941117	.080127
4.0000	2.305747	.955517	.064235
4.2000	2.498040	.966956	.050521
4.4000	2.692361	.975869	.038974
4.6000	2.888248	.982682	.029485
4.8000	3.085321	.987788	.021873
5.0000	3.283274	.991540	.015909
5.2000	3.481868	.994244	.011344
5.4000	3.680919	.996154	.007929
5.6000	3.880291	.997476	.005434
5.8000	4.079882	.998374	.003650
6.0000	4.279621	.998972	.002403
6.2000	4.479457	.999361	.001551
6.4000	4.679356	.999611	.000982
6.6000	4.879295	.999767	.000609
6.8000	5.079259	.999863	.000370
7.0000	5.279238	.999921	.000221
7.2000	5.479226	.999955	.000129
7.4000	5.679219	.999975	.000074
7.6000	5.879215	.999986	.000041
7.8000	6.079213	.999992	.000023
8.0000	6.279212	.999996	.000012
8.2000	6.479211	.999998	.000007
8.4000	6.679211	.999999	.000003
8.6000	6.879211	.999999	.000002
8.8000	7.079211	.999999	.000001
9.0000	7.279211	.999999	.000000
9.2000	7.479210	.999999	.000000
9.4000	7.679210	.999999	.000000
9.6000	7.879210	.999999	.000000
9.8000	8.079210	.999999	.000000
10.0000	8.279210	.999999	.000000

The results in table 1 may be visualized with the aid of figure 48. This plot represents the non-dimensional velocity ($F' = u/U_\infty$) profile in the boundary layer verses the similarity variable, η and shows the asymptotic nature of the solution. From an engineering standpoint, the similarity solution is important not only because it provides information on the extent of the boundary layer

(i.e. boundary layer thickness), but it also allows for the determination of skin friction (i.e. viscous drag). The drag can be obtained from the velocity distribution since for a Newtonian fluid, the shear stress at the plate surface (i.e. $\eta = 0$) is defined as

$$\tau_0 = \mu(\partial u / \partial y)|_{y=0}.$$

Thus, substituting $f'(0)$ into the above equation and integrating over the length of the plate gives the average shear stress or drag per unit width, an important parameter in hydrodynamics. It is convenient to put this result in non-dimensional form giving the local skin friction coefficient for the plate. This non-dimensional parameter may be defined as

$$C_f = \mu(\partial u / \partial y)|_{y=0}/\frac{1}{2}(\rho U_\infty^2).$$

Using the average shear stress obtained by integrating the local shear stress over the length of the plate in the above equation gives the average skin friction coefficient.

Finally, the results of the similarity solution may be used to determine two other important parameters, the displacement thickness, δ_d ; and the momentum thickness, δ_m . The physical significance of the displacement thickness is that it provides an indication of the degree to which the laminar streamlines of the external potential flow field are displaced as a result of viscous effects produced by the fluid interaction with the flat plate. The displacement thickness may be obtained by integrating $(1 - u/U_\infty)$ over the boundary layer thickness, $y = 0$ to ∞ . Alternatively, the momentum thickness indicates the reduction in the momentum of the fluid in the boundary layer as compared with the external potential flow. The momentum thickness is obtained by integrating $u(U_\infty - u)$ over the boundary layer thickness, $y = 0$ to ∞ . In both cases, the non-dimensional velocity u/U_∞ is obtained from the similarity solution as F' and substituted into the integrals with the appropriate transformation of the limits and integrated accordingly.

Part 3: The Thermal Boundary Layer Similarity Solution

As previously mentioned, having the velocity boundary layer similarity solution allows for the solution of the thermal boundary layer equation. This is achieved by substitution of F and F'' into the thermal boundary layer equation and then solving this equation using an appropriate matrix method. Physically, this problem represents a heated plate subjected to forced-convection. The solution was obtained for different values of Prandtl number, $Pr = 1.0, 0.714$ (air), and 6.75 (water) and is included as table 2. In table 2, the similarity variable, η , is included with the non-dimensional excess temperature defined as $\theta = (T - T_\infty)/(T_w - T_\infty)$. It is interesting to note the changes in the thermal boundary layer thickness as a result of changing Pr . Noting the asymptotic nature of the solution, the boundary layer thickness may be obtained as that value of y for which $\theta \rightarrow 0$ to some reasonable order, say 10^{-6} . Then, it can be seen that for higher values of Pr the thermal boundary layer thickness is smaller as compared with lower values of Pr . Thus, water ($Pr=6.75$) exhibits a much smaller thermal boundary layer thickness as compared with air ($Pr=0.714$). This is intuitively correct since Pr is defined as the ratio of kinematic viscosity, v to thermal diffusivity, α and provides an indication of the relative effects of momentum diffusivity and thermal diffusivity in the heat transfer process. Thus, since water has a relatively lower thermal diffusivity the predominant heat transfer process is convection and heat tends to be convected away from the surface of the plate. Alternatively, for air the predominant heat transfer process is conduction as heat tends to diffuse through the boundary layer faster than is convected away from the surface.

Another important parameter obtained from this solution is the recovery factor that provides the value of θ at the surface of the plate ($\eta = 0$). From a heat transfer standpoint this result gives the necessary information for determining the rate of heat transfer from the plate to the fluid. Applying Fourier's Law,

$$q''(x) = -k(\partial T / \partial y)|_{y=0}.$$

Where, q'' is the heat transfer per unit area (i.e. heat flux) and k is the thermal conductivity of the fluid. It can be shown by similarity transformation that this is equivalent to

$$q''(x) = -k(U_\infty/vx)^{1/2}(\partial T / \partial \eta)|_{\eta=0}.$$

This result can then be integrated over the length of the plate to obtain an expression for the total heat transfer rate, q' per unit width. Finally, it is convenient to put this result in non-dimensional form as the overall Nusselt number, defined as

$$\underline{\text{Nu}}_x = q'x/k\Delta T = \underline{h}_x x/k.$$

Where, \underline{h}_x is the average heat transfer coefficient.

Table 2. – Thermal boundary layer similarity solution.

Pr = 1.0		Pr = 0.714 (air)		Pr = 6.75 (water)	
η	θ	η	θ	η	θ
.000	1.000834	.000	.844722	.000	2.492125
.200	.996424	.200	.841573	.200	2.462354
.400	.983202	.400	.832131	.400	2.373335
.600	.961238	.600	.816438	.600	2.227093
.800	.930714	.800	.794601	.800	2.028960
1.000	.891982	1.000	.766829	1.000	1.788649
1.200	.845609	1.200	.733460	1.200	1.520416
1.400	.792402	1.400	.694976	1.400	1.241963
1.600	.733424	1.600	.652017	1.600	.972098
1.800	.669978	1.800	.605375	1.800	.727666
2.000	.603566	2.000	.555977	2.000	.520681
2.200	.535822	2.200	.504847	2.200	.356663
2.400	.468423	2.400	.453064	2.400	.234731
2.600	.402987	2.600	.401705	2.600	.149263
2.800	.340975	2.800	.351789	2.800	.092325
3.000	.283603	3.000	.304223	3.000	.055894
3.200	.231774	3.200	.259759	3.200	.033251
3.400	.186051	3.400	.218966	3.400	.019453
3.600	.146651	3.600	.182214	3.600	.011171
3.800	.113482	3.800	.149686	3.800	.006275
4.000	.086196	4.000	.121386	4.000	.003436
4.200	.064256	4.200	.097172	4.200	.001829
4.400	.047006	4.400	.076791	4.400	.000944
4.600	.033742	4.600	.059904	4.600	.000472
4.800	.023765	4.800	.046129	4.800	.000229
5.000	.016421	5.000	.035063	5.000	.000107
5.200	.011132	5.200	.026306	5.200	.000048

5.400	.007402	5.400	.019479	5.400	.000021
5.600	.004827	5.600	.014235	5.600	.000009
5.800	.003087	5.800	.010265	5.800	.000004
6.000	.001936	6.000	.007304	6.000	.000001
6.200	.001191	6.200	.005128	6.200	.000001
6.400	.000718	6.400	.003551	6.400	.000000
6.600	.000424	6.600	.002426	6.600	.000000
6.800	.000245	6.800	.001635	6.800	.000000
7.000	.000139	7.000	.001086	7.000	.000000
7.200	.000077	7.200	.000712	7.200	.000000
7.400	.000042	7.400	.000460	7.400	.000000
7.600	.000022	7.600	.000293	7.600	.000000
7.800	.000012	7.800	.000184	7.800	.000000
8.000	.000006	8.000	.000114	8.000	.000000
8.200	.000003	8.200	.000069	8.200	.000000
8.400	.000001	8.400	.000042	8.400	.000000
8.600	.000001	8.600	.000024	8.600	.000000
8.800	.000000	8.800	.000014	8.800	.000000
9.000	.000000	9.000	.000008	9.000	.000000
9.200	.000000	9.200	.000004	9.200	.000000
9.400	.000000	9.400	.000002	9.400	.000000
9.600	.000000	9.600	.000001	9.600	.000000
9.800	.000000	9.800	.000000	9.800	.000000
10.000	.000000	10.000	.000000	10.000	.000000

The results have also been presented as figure 49, a plot of the normalized non-dimensional excess temperature function, θ/θ_0 verses the similarity variable, η for each case of Pr investigated. The upper relationship represents the Pr=0.714 (air), while the lower curve is that for Pr=6.75(water). The relative sizes of the thermal boundary layers and the respective temperature gradients for each case can be visualized from these results.

CONCLUSIONS

- The results of this study indicate that although algorithm RKDUMB appears to be less efficient than the algorithms RKQS and BSSTEP, it provides superior results when looking for solution trajectory structure as in the case of the modified Lorenz set of ODEs. In the cases of RKQS and BSSTEP the “smoothness” of the solution trajectory is lost making RKDUMB the favored routine for this study.
- RKDUMB in combination with interval halving provides an effective solution technique for the two-point BVP, Blasius’ equation and the corresponding boundary conditions. These results compare well with those obtained by Howarth and presented by Schlichting, Boundary Layer Theory⁴. This is as expected since Howarth also used the fourth-order Runge-Kutta method.
- The matrix method algorithm TRIDAG is an effective method for solving second order ODEs as in the case of the thermal boundary layer similarity equation having previously obtained the velocity boundary layer similarity solution. The important feature of the TRIDAG algorithm is the recovery factor that provides the solution to the excess temperature at the surface of the plate. This result has physical significance as it can be used directly in obtaining the local and overall heat transfer characteristics of the problem.

REFERENCES

1. Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., Numerical Recipes in Fortran 77, 2nd Edition, Volume 1, Cambridge University Press, 1992.
2. Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., Numerical Recipes in Fortran 77 – Examples Book, Cambridge University Press, 1992.
3. Chapra, S.C., and Canale, R.C., Introduction to Computing for Engineers, McGraw-Hill Book Co., 1986.
4. Schlichting, H., Boundary Layer Theory, 7th Edition, McGraw-Hill, 1979.
5. Bejan, A., Heat Transfer, John Wiley & Sons, 1993.

APPENDIX B – FORTRAN CODE

Problem 1a_1.) Computes the solution to the modified Lorenz set of differential equations. The driver program ***hw3pr1a1.f*** makes use of numerical recipes, ***rkdumb.f***, an extension of ***rk4.f*** that implements the fourth-order Runge-Kutta algorithm. The subroutine ***derives1.f*** was written for the Lorenz set.

```
PROGRAM hw3pr1a1
INTEGER nstep,nvar
PARAMETER (nvar=3,nstep=60000)
INTEGER i,j
REAL xi,yi,zi,x(60001),x1,x2,vstart(nvar)
EXTERNAL derivs1
common /path/ xx(60001),y(10,60001)
xi=-6.
yi=-6.
zi=2.
x1=0.0
x2=300.
vstart(1)=xi
vstart(2)=yi
vstart(3)=zi
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs1)
OPEN(unit=45,file='hw3pr1a1.out',status='unknown')
write(45,*)' time',' x',' y',' z'
do 11 i=1,nstep
write(45,'(1x,f10.4,2x,3f12.6)')xx(i),y(1,i),y(2,i),y(3,i)
11 continue
END
```

```
subroutine rkdumb(vstart,nvar,x1,x2,nstep,derivs)
```

```
parameter (nmax=3)
common /path/ xx(60001),y(3,60001)
dimension vstart(nvar),v(nmax),dv(nmax)
external derivs
```

```
C
C To run rkbumb you need to have the subroutine derivs written
C and the subroutine rk4 in the correct directory. See information
C concerning make files on the plot directory. Also note the use
C of a common block here, you need to have the common block in
C your main program also, if you are not sure about the syntax
C consult a fortran book.
```

```
C
```

```
C INPUT
```

```
C vstart--> the initial conditions of the n variables
C nvar --> the number of variable you wish to integrate(3)
C x1 --> the starting value for the integration(0.0)
C x2 --> the ending value for the integration(7.0)
C nstep --> the number of steps to take from x1 to x2(35)
C derivs--> a subroutine to find RHS values
```

```
C
```

```
C OUTPUT
```

```
C All output is stored in the common block where
```

```

C   xx is the list of steps, namely eta from 0 to 7 by .2 steps,
C   and y is the list of function values for each step, where
C   y will be 35 elements long like xx but each element will have
C   three values, one for F, F', and F".
C
C   vstart is an array, in our case, three elements long which needs
C   to contain the initial condition of F, F', and F".
C
C   derivs is a subroutine which when call returns the value of the
C   right hand sides of the three derivative equations. The subroutine
C   needs to be written by you and store the three RHS values in an
C   array called dydx.
C
do 11 i=1,nvar
  v(i)=vstart(i)
  y(i,1)=v(i)
11 continue
  xx(1)=x1
  x=x1
  h=(x2-x1)/nstep
  do 13 k=1,nstep
    v(i)=vstart(i)
    y(i,1)=v(i)
11 continue
  xx(1)=x1
  x=x1
  h=(x2-x1)/nstep
  do 13 k=1,nstep
    call derivs(x,v,dv)
    call rk4(v,dv,nvar,x,h,v,derivs)
    if(x+h.eq.x)pause 'stepsize not significant in rkdumb.'
    x=x+h
    xx(k+1)=x
    do 12 i=1,nvar
      y(i,k+1)=v(i)
12 continue
13 continue
  return
end

subroutine rk4(y,dydx,n,x,h,yout,derivs)
parameter (nmax=3)
dimension y(n),dydx(n),yout(n),yt(nmax),dyt(nmax),dym(nmax)
external derivs
C
C   INPUT
C   y --> an array of the values of F, F', and F" at the current step
C   dydx --> an array of the derivatives of F, F', and F"(found using
C           the derivs subroutine)
C   n --> the number of variables we wish to solve for.(3)
C   x --> the x position to solve,we are using eta
C   h --> the step size, will be computed in rkdumb
C

```

```

C  OUTPUT
C      yout --> an array of the values of F,F',and F" at the next step
C      derivs-> subroutine see explanation in rkdumb
C

hh=h*0.5
h6=h/6.
xh=x+hh
do 11 i=1,n
    yt(i)=y(i)+hh*dydx(i)
11 continue
call derivs(xh,yt,dyt)
do 12 i=1,n
    yt(i)=y(i)+hh*dyt(i)
12 continue
call derivs(xh,yt,dym)
do 13 i=1,n
    yt(i)=y(i)+h*dym(i)
    dym(i)=dyt(i)+dym(i)
13 continue
call derivs(x+h,yt,dyt)
do 14 i=1,n
    yout(i)=y(i)+h6*(dydx(i)+dyt(i)+2.*dym(i))
14 continue
return
end

subroutine derivs1(x,y,dydx)
REAL x,y(*),dydx(*)
dydx(1)=10.*(y(2)-y(1))
dydx(2)=y(1)*(28.-y(3))-y(2)
dydx(3)=y(1)*y(2)-8.*y(3)/3.
return
END

```

*eddie> f77 hw3pr1a1.f rkdumb.f rk4.f derivs1.f -o hw3p1a1
output fn=hw3pr1a1.out*

Problem 1a_2.) Computes the solution to the modified Lorenz set of differential equations. The driver program **hw3pr1a2.f** makes use of numerical recipes, **odeint.f**, that makes use of **rkqs.f**, an adaptive step-size algorithm implementing **rkck.f**, the fifth-order Runge-Kutta algorithm. The subroutine **derives2.f** was written for the Lorenz set to provide information on the number of function evaluations.

```

PROGRAM hw3pr1a2
INTEGER n
PARAMETER (n=3)
INTEGER kmaxx,nmax,nvar
PARAMETER (kmaxx=60001,nmax=3,nvar=3)
INTEGER i,kmax,kount,nbad,nok,nrhs
REAL xi,yi,zi
REAL dxsav,eps,h1,hmin,x1,x2,x,y,ystart(nvar)

```

```

COMMON /path/kmax,kount,dxsav,x(kmaxx),y(nmax,kmaxx)
COMMON nrhs
EXTERNAL derivs2
nrhs=0
x1=0.0
x2=300.
xi=-6.
yi=-6.
zi=2.
ystart(1)=xi
ystart(2)=yi
ystart(3)=zi
eps=1.0e-3
h1=0.005
hn=0.0
kmax=60001
dxsav=(x2-x1)/60000
CALL odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs2,rkqs)
OPEN(unit=45,file='hw3pr1a2.out',status='unknown')
write(45,'(1x,a,t30,i10)'# function evaluations:',nrhs
write(45,*) time', x', y', z'
do 11 i=1,kount
    write(45,'(1x,f10.4,2x,3f12.6)'x(i),y(1,i),y(2,i),y(3,i)
11 continue
END

SUBROUTINE odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs,
*rkqs)
INTEGER nbad,nok,nvar,KMAXX,MAXSTP,NMAX
REAL eps,h1,hmin,x1,x2,ystart(nvar),TINY
EXTERNAL derivs,rkqs
PARAMETER (MAXSTP=10000,NMAX=3,KMAXX=60000,TINY=1.e-30)
INTEGER i,kmax,kount,nstp
REAL dxsav,h,hdid,hnext,x,xsav,dydx(NMAX),xp(KMAXX),y(NMAX),
*yp(NMAX,KMAXX),yscal(NMAX)
COMMON /path/ kmax,kount,dxsav,xp,yp
x=x1
h=sign(h1,x2-x1)
nok=0
nbad=0
kount=0
do 11 i=1,nvar
    y(i)=ystart(i)
11 continue
if (kmax.gt.0) xsav=x-2.*dxsav
do 16 nstp=1,MAXSTP
    call derivs(x,y,dydx)
    do 12 i=1,nvar
        yscal(i)=abs(y(i))+abs(h*dydx(i))+TINY
12 continue
if(kmax.gt.0)then
    if(abs(x-xsav).gt.abs(dxsav)) then
        if(kount.lt.kmax-1)then

```

```

        kount=kount+1
        xp(kount)=x
        do 13 i=1,nvar
          yp(i,kount)=y(i)
13      continue
        xsav=x
        endif
        endif
        endif
        if((x+h-x2)*(x+h-x1).gt.0.) h=x2-x
        call rkqs(y,dydx,nvar,x,h,eps,yscal,hdid,hnext,derivs)
        if(hdid.eq.h)then
          nok=nok+1
        else
          nbad=bad+1
        endif
        if((x-x2)*(x2-x1).ge.0.)then
          do 14 i=1,nvar
            ystart(i)=y(i)
14      continue
        if(kmax.ne.0)then
          kount=kount+1
          xp(kount)=x
          do 15 i=1,nvar
            yp(i,kount)=y(i)
15      continue
        endif
        return
        endif
        if(abs(hnext).lt.hmin) pause
      *'stepsize smaller than minimum in odeint'
        h=hnext
16  continue
        pause 'too many steps in odeint'
        return
      END

SUBROUTINE rkqs(y,dydx,n,x,htry,eps,yscal,hdid,hnext,derivs)
INTEGER n,NMAX
REAL eps,hdid,hnext,htry,x,dydx(n),y(n),yscal(n)
EXTERNAL derivs
PARAMETER (NMAX=3)
INTEGER i
REAL errmax,h,xnew,yerr(NMAX),ytemp(NMAX),SAFETY,PGROW,
* PSHRNK,ERRCON
PARAMETER (SAFETY=0.9,PGROW=-.2,PSHRNK=-.25,ERRCON=1.89E-4)
h=htry
1  call rkck(y,dydx,n,x,h,ytemp,yerr,derivs)
errmax=0
do 100 i=1,n
  errmax=max(errmax,abs(yerr(i)/yscal(i)))
100 continue
errmax=errmax/eps

```

```

if(errmax.gt.1.)then
  h=SAFETY*h*(errmax**PSHRNK)
  if(h.lt.0.1*h)then
    h=.1*h
  endif
  xnew=x+h
  if(xnew.eq.x)pause 'stepsize underflow in rkqd'
  goto 1
else
  if(errmax.gt.ERRCON)then
    hnnext=SAFETY*h*(errmax**PGROW)
  else
    hnnext=5.*h
  endif
  hdid=h
  x=x+h
  do 101 i=1,n
    y(i)=ytemp(i)
101  continue
  return
endif
END

SUBROUTINE rkck(y,dydx,n,x,h,yout,yerr,derivs)
INTEGER n,NMAX
REAL h,x,dydx(n),y(n),yerr(n),yout(n)
EXTERNAL derivs
PARAMETER (NMAX=3)
INTEGER i
REAL ak2(NMAX),ak3(NMAX),ak4(NMAX),ak5(NMAX),ak6(NMAX),
*   ytemp(NMAX),A2,A3,A4,A5,A6,B21,B31,B32,B41,B42,B43,B51,
*   B52,B53,B54,B61,B62,B63,B64,B65,C1,C3,C4,C6,DC1,DC3,
*   DC4,DC5,DC6
PARAMETER (A2=.2,A3=.3,A4=.6,A5=1.,A6=.875,B21=.2,B31=.40.,
*   B32=.40.,B41=.3,B42=-.9,B43=1.2,B51=-11./54.,B52=2.5,
*   B53=-70./27.,B54=35./27.,B61=1631./55296.,B62=175./512.,
*   B63=575./13824.,B64=44275./110592.,B65=253./4096.,
*   C1=37./378.,C3=250./621.,C4=125./594.,C6=512./1771.,
*   DC1=C1-2825./27648.,DC3=C3-18575./48384.,
*   DC4=C4-13525./55296.,DC5=-277./14336.,DC6=C6-.25)
do 100 i=1,n
  ytemp(i)=y(i)+B21*h*dydx(i)
100  continue
  call derivs(x+A2*h,ytemp,ak2)
  do 101 i=1,n
    ytemp(i)=y(i)+h*(B31*dydx(i)+B32*ak2(i))
101  continue
  call derivs (x+A3*h,ytemp,ak3)
  do 102 i=1,n
    ytemp(i)=y(i)+h*(B41*dydx(i)+B42*ak2(i)+B43*ak3(i))
102  continue
  call derivs(x+A4*h,ytemp,ak4)
  do 103 i=1,n

```

```

        ytemp(i)=y(i)+h*(B51*dydx(i)+B52*ak2(i)+B53*ak3(i) +
      *   B54*ak4(i))
103  continue
      call derivs(x+A5*h,ytemp,ak5)
      do 104 i=1,n
          ytemp(i)=y(i)+h*(B61*dydx(i)+B62*ak2(i)+B63*ak3(i) +
      *   B64*ak4(i)+B65*ak5(i))
104  continue
      call derivs(x+A6*h,ytemp,ak6)
      do 105 i=1,n
          yout(i)=y(i)+h*(C1*dydx(i)+C3*ak3(i)+C4*ak4(i) +
      *   C6*ak6(i))
105  continue
      do 106 i=1,n
          yell(i)=h*(DC1*dydx(i)+DC3*ak3(i)+DC4*ak4(i)+DC5*ak5(i) +
          DC6*ak6(i))
106  continue
      RETURN
      END

```

```

SUBROUTINE derivs2(x,y,dydx)
REAL x,y(*),dydx(*)
INTEGER nrhs
COMMON nrhs
nrhs=nrhs+1
dydx(1)=10.*(y(2)-y(1))
dydx(2)=y(1)*(28.-y(3))-y(2)
dydx(3)=y(1)*y(2)-8.*y(3)/3.
return
END

```

```

eddie> f77 hw3pr1a2.f derivs2.f odeint.f rkqs.f rkck.f -o h3p1a2
output fn=hw3pr1a2.out
# function evaluations:      24318

```

Problem 1a_3.) Computes the solution to the modified Lorenz set of differential equations. The driver program *hw3pr1a3.f* makes use of numerical recipes, *odeint.f*, that makes use of *bsstep.f* (similar to *rkqs.f*) an adaptive step-size algorithm that performs successive interpolation using *mmid.f*, and *rzextr.f*. The subroutine *derives2.f* was written for the Lorenz set to provide information on the number of function evaluations.

```

PROGRAM hw3pr1a3
INTEGER n
PARAMETER (n=3)
INTEGER kmaxx,nmax,nvar
PARAMETER (kmaxx=60001,nmax=3,nvar=3)
INTEGER i,kmax,kount,nbad,nok,nrhs
REAL xi,yi,zi
REAL dxsav,eps,h1,hmin,x1,x2,x,y,ystart(nvar)
COMMON /path/kmax,kount,dxsav,x(kmaxx),y(nmax,kmaxx)
COMMON nrhs
EXTERNAL derivs2,bsstep

```

```

nrhs=0
x1=0.0
x2=300.
xi=-6.
yi=-6.
zi=2.
ystart(1)=xi
ystart(2)=yi
ystart(3)=zi
eps=1.0e-3
h1=0.005
hn=0.0
kmax=60001
dxsav=(x2-x1)/60000
CALL odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs2,bsstep)
OPEN(unit=45,file='hw3pr1a3.out',status='unknown')
write(45,'(1x,a,t30,i10)')# function evaluations:',nrhs
write(45,*)' time',' x',' y',' z'
do 11 i=1,kount
    write(45,'(1x,f10.4,2x,3f12.6)')x(i),y(1,i),y(2,i),y(3,i)
11 continue
END

SUBROUTINE odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs2,
*bsstep)
INTEGER nbad,nok,nvar,KMAXX,MAXSTP,NMAX
REAL eps,h1,hmin,x1,x2,ystart(nvar),TINY
EXTERNAL derivs,bsstep
PARAMETER (MAXSTP=10000,NMAX=3,KMAXX=60000,TINY=1.e-30)
INTEGER i,kmax,kount,nstp
REAL dxsav,h,hdid,hnext,x,xsav,dydx(NMAX),xp(KMAXX),y(NMAX),
*yp(NMAX,KMAXX),yscal(NMAX)
COMMON /path/ kmax,kount,dxsav,xp,yp
x=x1
h=sign(h1,x2-x1)
nok=0
nbad=0
kount=0
do 11 i=1,nvar
    y(i)=ystart(i)
11 continue
if (kmax.gt.0) xsav=x-2.*dxsav
do 16 nstp=1,MAXSTP
    call derivs(x,y,dydx)
    do 12 i=1,nvar
        yscal(i)=abs(y(i))+abs(h*dydx(i))+TINY
12 continue
if(kmax.gt.0)then
    if(abs(x-xsav).gt.abs(dxsav)) then
        if(kount.lt.kmax-1)then
            kount=kount+1
            xp(kount)=x
        do 13 i=1,nvar

```

```

      yp(i,kount)=y(i)
13    continue
      xsav=x
      endif
      endif
      endif
      if((x+h-x2)*(x+h-x1).gt.0.) h=x2-x
      call bsstep(y,dydx,nvar,x,h,eps,yscal,hdid,hnext,derivs2)
      if(hdid.eq.h)then
        nok=nok+1
      else
        nbad=nbad+1
      endif
      if((x-x2)*(x2-x1).ge.0.)then
        do 14 i=1,nvar
          ystart(i)=y(i)
14    continue
      if(kmax.ne.0)then
        kount=kount+1
        xp(kount)=x
        do 15 i=1,nvar
          yp(i,kount)=y(i)
15    continue
      endif
      return
      endif
      if(abs(hnext).lt.hmin) pause
      *'stepsize smaller than minimum in odeint'
      h=hnext
16 continue
      pause 'too many steps in odeint'
      return
      END

subroutine bsstep(y,dydx,nv,x,htry,eps,yscal,hdid,hnext,derivs)
parameter (nmax=3,imax=11,nuse=7,one=1.e0,shrink=.95e0,grow=1.2e0
*)
dimension y(nv),dydx(nv),yscal(nv),yerr(nmax),
*   ysav(nmax),dysav(nmax),yseq(nmax),nseq(imax)
C
C   NOTE: same call list as rkqc!!!!!(nv = n)
C
C   INPUT
C     y --> an array of starting values, initial conditions
C     dydx-> an array of derivatives at eh initial conditions
C     nv --> the number of variables being solved for
C     x --> the starting value of the integration(0.0)
C     htry-> the step size to attempt
C     eps -> the allowable error(10E-6)
C     yscal-> an array of against which error is scaled
C
C   OUTPUT
C     hdid-> the step size that was accomplished

```

```

C   hnext-> an estimate of the next step size
C   derivs-> RHS evaluator, see rkdumb
C   y and x also act as output values therefore on each call to
C   rkqc y and x are replaced with the values at the next time step.
C
C   Try different scales for the array yscal, note that after
C   the first guess on htry, the next one will always be given
C   in the form of hnext from the previous call.

data nseq /2,4,6,8,12,16,24,32,48,64,96/
h=htry
xsav=x
do 11 i=1,nv
  ysav(i)=y(i)
  dysav(i)=dydx(i)
11 continue
1  do 10 i=1,imax
    call mmid(ysav,dysav,nv,xsav,h,nseq(i),yseq,derivs)
    xest=(h/nseq(i))**2
    call rzextr(i,xest,yseq,y,yerr,nv,nuse)
    errmax=0.
    do 12 j=1,nv
      errmax=max(errmax,abs(yerr(j)/yscal(j)))
12 continue
errmax=errmax/eps
if(errmax.lt.one) then
  x=x+h
  hdid=h
  if(i.eq.nuse)then
    hnext=h*shrink
  else if(i.eq.nuse-1)then
    hnext=h*grow
  else
    hnext=(h*nseq(nuse-1))/nseq(i)
  endif
  return
endif
10 continue
h=0.25*h/2**((imax-nuse)/2)
if(x+h.eq.x)pause 'step size underflow.'
goto 1
end

subroutine mmid(y,dydx,nvar,xs,htot,nstep,yout,derivs)
parameter (nmax=3)
dimension y(nvar),dydx(nvar),yout(nvar),ym(nmax),yn(nmax)
C
C   mmid is called by bsstep and all variables needed are
C   already defined by bsstep.
C
h=htot/nstep
do 11 i=1,nvar
  ym(i)=y(i)

```

```

      yn(i)=y(i)+h*dydx(i)
11  continue
      x=xs+h
      call derivs(x,yn,yout)
      h2=2.*h
      do 13 n=2,nstep
          do 12 i=1,nvar
              swap=ym(i)+h2*yout(i)
              ym(i)=yn(i)
              yn(i)=swap
12  continue
      x=x+h
      call derivs(x,yn,yout)
13  continue
      do 14 i=1,nvar
          yout(i)=0.5*(ym(i)+yn(i)+h*yout(i))
14  continue
      return
      end

subroutine rzextr(iest,xest,yest,yz,dy,nv,nuse)
parameter (imax=11,nmax=10,ncol=7)
dimension x(imax),yest(nv),yz(nv),dy(nv),d(nmax,ncol),fx(ncol)
C
C   rzextr is called by bsstep and all variables needed are
C   already defined by bsstep.
C
      x(iest)=xest
      if(iest.eq.1) then
          do 11 j=1,nv
              yz(j)=yest(j)
              d(j,1)=yest(j)
              dy(j)=yest(j)
11  continue
      else
          m1=min(iest,nuse)
          do 12 k=1,m1-1
              fx(k+1)=x(iest-k)/xest
12  continue
      do 14 j=1,nv
          yy=yest(j)
          v=d(j,1)
          c=yy
          d(j,1)=yy
          do 13 k=2,m1
              b1=fx(k)*v
              b=b1-c
              if(b.ne.0.) then
                  b=(c-v)/b
                  ddy=c*b
                  c=b1*b
              else
                  ddy=v
13  continue
14  continue
      end

```

```

        endif
        v=d(j,k)
        d(j,k)=ddy
        yy=yy+ddy
13    continue
        dy(j)=ddy
        yz(j)=yy
14    continue
endif
return
end

```

```

SUBROUTINE derivs2(x,y,dydx)
REAL x,y(*),dydx(*)
INTEGER nrhs
COMMON nrhs
nrhs=nrhs+1
dydx(1)=10.*(y(2)-y(1))
dydx(2)=y(1)*(28.-y(3))-y(2)
dydx(3)=y(1)*y(2)-8.*y(3)/3.
return
END

```

```

eddie> f77 hw3pr1a3.f odeint.f bsstep.f mmid.f rzextr.f derivs2.f -o h3p1a3
output fn=hw3pr1a3.out
# function evaluations:      71159

```

Problem 1b_1.) Computes the solution to the Lorenz set of differential equations for case $\theta=60^\circ$. The driver program ***hw3pr1b1.f*** makes use of numerical recipes, ***rkdumb.f***, an extension of ***rk4.f*** that implements the fourth-order Runge-Kutta algorithm. The subroutine ***derives3.f*** was written for the Lorenz set with the perturbation terms f and θ .

```

PROGRAM hw3pr1b1
INTEGER nstep,nvar
PARAMETER (nvar=3,nstep=60000)
INTEGER i,j
REAL xi,yi,zi,x(60001),x1,x2,vstart(nvar)
EXTERNAL derivs3
common /path/ xx(60001),y(3,60001)
xi=-6.
yi=-6.
zi=2.
x1=0.0
x2=300.
vstart(1)=xi
vstart(2)=yi
vstart(3)=zi
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs3)
OPEN(unit=45,file='hw3pr1b1.out',status='unknown')
write(45,*)' time',' x',' y',' z'
do 11 i=1,nstep
  write(45,'(1x,f10.4,2x,3f12.6)')xx(i),y(1,i),y(2,i),y(3,i)
11   continue

```

```

11 continue
END

SUBROUTINE derivs3(x,y,dydx)
REAL x,y(*),dydx(*)
dydx(1)=10.*(y(2)-y(1))+2.5*cos(60*3.14159/180.0)
dydx(2)=y(1)*(28.-y(3))-y(2)+2.5*sin(60*3.14159/180.0)
dydx(3)=y(1)*y(2)-8.*y(3)/3.
return
END

```

*eddie> f77 hw3pr1b1.f rkdumb.f rk4.f derivs3.f -o h3p1b1
output fn=hw3pr1b1.out*

Problem 1b_2.) Computes the solution to the Lorenz set of differential equations for case $\theta=240^\circ$. The driver program ***hw3pr1b2.f*** makes use of numerical recipes, ***rkdumb.f***, an extension of ***rk4.f*** that implements the fourth-order Runge-Kutta algorithm. The subroutine ***derives4.f*** was written for the Lorenz set with the perturbation terms f and θ .

```

PROGRAM hw3pr1b2
INTEGER nstep,nvar
PARAMETER (nvar=3,nstep=60000)
INTEGER i,j
REAL xi,yi,zi,x(60001),x1,x2,vstart(nvar)
EXTERNAL derivs4
common /path/ xx(60001),y(3,60001)
xi=-6.
yi=-6.
zi=2.
x1=0.0
x2=300.
vstart(1)=xi
vstart(2)=yi
vstart(3)=zi
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs4)
OPEN(unit=45,file='hw3pr1b2.out',status='unknown')
write(45,' time',' x',' y',' z'
do 11 i=1,nstep
    write(45,'(1x,f10.4,2x,3f12.6)')xx(i),y(1,i),y(2,i),y(3,i)
11 continue
END

```

```

SUBROUTINE derivs4(x,y,dydx)
REAL x,y(*),dydx(*)
dydx(1)=10.*(y(2)-y(1))+2.5*cos(240*3.14159/180.0)
dydx(2)=y(1)*(28.-y(3))-y(2)+2.5*sin(240*3.14159/180.0)
dydx(3)=y(1)*y(2)-8.*y(3)/3.
return
END

```

*eddie> f77 hw3pr1b2.f rkdumb.f rk4.f derivs4.f -o h3p1b2
output fn=hw3pr1b2.out*

Problem 2.) RKDUMB solution to Blasius' flat plate similarity equation using $f'(0)=0.0$, $f''(0)=0.0$, and $f'''(0)=0.2$ with $\Delta f'=0.05$. Interval halving was used in conjunction with *rkdumb.f*.

```

PROGRAM hw3pr2
INTEGER nstep
PARAMETER (eps=1.0e-6,nvar=3,nstep=50)
INTEGER i
REAL p,dq,xx(51),y(nvar,51),x1,x2,vstart(nvar),error
EXTERNAL derivs5
common /path/ xx,y
c Initial values
dq=0.05
q=0.2
10 x1=0.0
x2=10.0
vstart(1)=0.0
vstart(2)=0.0
q=q+dq
vstart(3)=q
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs5)
c Implement interval halving to improve starting values
error=1.0-y(2,50)
if (error.gt.0.0.and.error.le.eps) then
  go to 20
else
  if ((y(2,50).gt.1.0.and.dq.gt.0.0).or.(y(2,50).lt.1.0.and.
& dq.lt.0.0)) then
    dq=-dq/2.
    end if
    go to 10
  end if
20 OPEN(unit=45,file='hw3pr2a.out',status='unknown')
c write(45,*)' eta',' f',' f''',' f'''
do 11 i=1,(nstep+1)
  write(45,'(1x,f8.4,2x,3f10.6)')xx(i),y(1,i),y(2,i),y(3,i)
11 continue
END

```

```

subroutine rkdumb(vstart,nvar,x1,x2,nstep,derivs)
parameter (nmax=3)
common /path/ xx(51),y(nmax,51)
dimension vstart(nvar),v(nmax),dv(nmax)
external derivs

```

```

C
C To run rkbumb you need to have the subroutine derivs written
C and the subroutine rk4 in the correct directory. See information
C concerning make files on the plot directory. Also note the use
C of a common block here, you need to have the common block in
C your main program also, if you are not sure about the syntax
C consult a fortran book.
C

```

```

C INPUT
C vstart--> the initial conditions of the n variables
C nvar --> the number of variable you wish to integrate(3)
C x1 --> the starting value for the integration(0.0)
C x2 --> the ending value for the integration(7.0)
C nstep --> the number of steps to take from x1 to x2(35)
C derivs--> a subroutine to find RHS values
C
C OUTPUT
C All output is stored in the common block where
C xx is the list of steps, namely eta from 0 to 7 by .2 steps,
C and y is the list of function values for each step, where
C y will be 35 elements long like xx but each element will have
C three values, one for F, F', and F".
C
C
C
C vstart is an array, in our case, three elements long which needs
C to contain the initial condition of F, F', and F".
C
C derivs is a subroutine which when call returns the value of the
C right hand sides of the three derivative equations. The subroutine
C needs to be written by you and store the three RHS values in an
C array called dydx.
C
do 11 i=1,nvar
  v(i)=vstart(i)
  y(i,1)=v(i)
11 continue
  xx(1)=x1
  x=x1
  h=(x2-x1)/nstep
  do 13 k=1,nstep
    call derivs(x,v,dv)
    call rk4(v,dv,nvar,x,h,v,derivs)
    if(x+h.eq.x)pause 'stepsize not significant in rkdumb.'
    x=x+h
    xx(k+1)=x
    do 12 i=1,nvar
      y(i,k+1)=v(i)
12 continue
13 continue
  return
end

subroutine rk4(y,dydx,n,x,h,yout,derivs)
parameter (nmax=3)
dimension y(n),dydx(n),yout(n),yt(nmax),dyt(nmax),dym(nmax)
external derivs
C
C INPUT
C y --> an array of the values of F, F', and F" at the current step
C dydx --> an array of the derivatives of F, F', and F"(found using

```

```

C      the derivs subroutine)
C      n --> the number of variables we wish to solve for.(3)
C      x --> the x position to solve,we are using eta
C      h --> the step size, will be computed in rkdump
C
C      OUTPUT
C      yout --> an array of the values of F,F',and F'' at the next step
C      derivs-> subroutine see explanation in rkdump
C
C      hh=h*0.5
C      h6=h/6.
C      xh=x+hh
C      do 11 i=1,n
C          yt(i)=y(i)+hh*dydx(i)
11  continue
        call derivs(xh,yt,dyt)
        do 12 i=1,n
          yt(i)=y(i)+hh*dyt(i)
12  continue
        call derivs(xh,yt,dym)
        do 13 i=1,n
          yt(i)=y(i)+h*dym(i)
          dym(i)=dyt(i)+dym(i)
13  continue
        call derivs(x+h,yt,dyt)
        do 14 i=1,n
          yout(i)=y(i)+h6*(dydx(i)+dyt(i)+2.*dym(i))
14  continue
        return
      end

```

```

SUBROUTINE derivs5(x,y,dydx)
REAL x,y(*),dydx(*)
dydx(1)=y(2)
dydx(2)=y(3)
dydx(3)=-0.5*y(1)*y(3)
return
END

```

eddie> f77 hw3pr2.f rkdump.f rk4.f derivs5.f -o h3p2

OUTPUT: fn=hw3pr2.out

Eta	<u>f</u>	<u>f'</u>	<u>f''</u>
.0000	.000000	.000000	.332057
.2000	.006641	.066408	.331984
.4000	.026560	.132764	.331470
.6000	.059735	.198937	.330079
.8000	.106109	.264709	.327389
1.0000	.165572	.329780	.323007
1.2000	.237949	.393776	.316589
1.4000	.322982	.456261	.307865
1.6000	.420322	.516756	.296663
1.8000	.529519	.574758	.282931

2.0000	.650025	.629765	.266752
2.2000	.781194	.681310	.248351
2.4000	.922291	.728981	.228092
2.6000	1.072507	.772454	.206455
2.8000	1.230978	.811509	.184007
3.0000	1.396809	.846044	.161361
3.2000	1.569095	.876080	.139128
3.4000	1.746951	.901760	.117877
3.6000	1.929526	.923329	.098087
3.8000	2.116030	.941117	.080127
4.0000	2.305747	.955517	.064235
4.2000	2.498040	.966956	.050521
4.4000	2.692361	.975869	.038974
4.6000	2.888248	.982682	.029485
4.8000	3.085321	.987788	.021873
5.0000	3.283274	.991540	.015909
5.2000	3.481868	.994244	.011344
5.4000	3.680919	.996154	.007929
5.6000	3.880291	.997476	.005434
5.8000	4.079882	.998374	.003650
6.0000	4.279621	.998972	.002403
6.2000	4.479457	.999361	.001551
6.4000	4.679356	.999611	.000982
6.6000	4.879295	.999767	.000609
6.8000	5.079259	.999863	.000370
7.0000	5.279238	.999921	.000221
7.2000	5.479226	.999955	.000129
7.4000	5.679219	.999975	.000074
7.6000	5.879215	.999986	.000041
7.8000	6.079213	.999992	.000023
8.0000	6.279212	.999996	.000012
8.2000	6.479211	.999998	.000007
8.4000	6.679211	.999999	.000003
8.6000	6.879211	.999999	.000002
8.8000	7.079211	.999999	.000001
9.0000	7.279211	.999999	.000000
9.2000	7.479210	.999999	.000000
9.4000	7.679210	.999999	.000000
9.6000	7.879210	.999999	.000000
9.8000	8.079210	.999999	.000000
10.0000	8.279210	.999999	.000000

Problem 3.) Flat plate temperature similarity solution (**Pr=1.0**). The driver program **hw3pr3.f** is the modified boundary layer similarity solution used in problem 2. Thus, this program solves the entire similarity solution (i.e. velocity and temperature) for the flat plate case with Pr=1.0.

```

PROGRAM hw3pr3
INTEGER nstep
PARAMETER (eps=1.0e-6,nvar=3,nstep=50,Pr=1.0)
INTEGER i,j,k
REAL p,dq,xx(51),y(nvar,51),x1,x2,vstart(nvar),error
REAL h,u(51),a(51),b(51),c(51),r(51)
EXTERNAL derivs5
common /path/ xx,y

```

```

h=0.2
dq=0.05
q=0.2
10 x1=0.0
x2=10.0
vstart(1)=0.0
vstart(2)=0.0
q=q+dq
vstart(3)=q
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs5)
error=1.0-y(2,50)
if (error.gt.0.0.and.error.le.eps) then
    go to 20
else
    if ((y(2,50).gt.1.0.and.dq.gt.0.0).or.(y(2,50).lt.1.0.and.
& dq.lt.0.0)) then
        dq=-dq/2.
        end if
        go to 10
    end if
20 a(1)=1.0-h*Pr*y(1,1)/4.0
b(1)=-2.0
c(1)=1.0+h*Pr*y(1,1)/4.0+1-h*Pr*y(1,1)/4.0
r(1)=-2.0*Pr*h**2*y(3,1)**2
do 30 i=2,(nstep+1)
    a(i)=1.0-h*Pr*y(1,i)/4.0
    b(i)=-2.0
    r(i)=-2.0*Pr*h**2*y(3,i)**2
30 continue
do 40 j=2,nstep
    c(j)=1.0+h*Pr*y(1,j)/4.0
40 continue
CALL tridag(a,b,c,r,u,nstep)
OPEN(unit=45,file='hw3pr3a.out',status='unknown')
write(45,*)'Recovery Factor ='
write(45,'(5x,f10.6)')u(1)
write(45,*)' eta,' theta'
do 50 k=1,(nstep+1)
    write(45,'(1x,f8.4,3x,f10.6)')xx(k),u(k)
50 continue
END

```

```

subroutine tridag(a,b,c,r,u,n)
parameter (nmax=100)
dimension gam(nmax),a(n),b(n),c(n),r(n),u(n)
if(b(1).eq.0.)pause
bet=b(1)
u(1)=r(1)/bet
do 11 j=2,n
    gam(j)=c(j-1)/bet
    bet=b(j)-a(j)*gam(j)
    if(bet.eq.0.)pause
    u(j)=(r(j)-a(j)*u(j-1))/bet
11 continue
end

```

```

11 continue
do 12 j=n-1,1,-1
  u(j)=u(j)-gam(j+1)*u(j+1)
12 continue
return
end

SUBROUTINE derivs5(x,y,dydx)
REAL x,y(*),dydx(*)
dydx(1)=y(2)
dydx(2)=y(3)
dydx(3)=-0.5*y(1)*y(3)
return
END

```

eddie> f77 hw3pr3.f rkdumb.f rk4.f derivs2a.f tridag.f -o h3p3

OUTPUT: fn=hw3pr3.out

Recovery Factor, $\theta(\eta=0)$ =

1.000834

<u>eta</u>	<u>theta</u>
.000	1.000834
.200	.996424
.400	.983202
.600	.961238
.800	.930714
1.000	.891982
1.200	.845609
1.400	.792402
1.600	.733424
1.800	.669978
2.000	.603566
2.200	.535822
2.400	.468423
2.600	.402987
2.800	.340975
3.000	.283603
3.200	.231774
3.400	.186051
3.600	.146651
3.800	.113482
4.000	.086196
4.200	.064256
4.400	.047006
4.600	.033742
4.800	.023765
5.000	.016421
5.200	.011132
5.400	.007402
5.600	.004827
5.800	.003087
6.000	.001936
6.200	.001191

6.400	.000718
6.600	.000424
6.800	.000245
7.000	.000139
7.200	.000077
7.400	.000042
7.600	.000022
7.800	.000012
8.000	.000006
8.200	.000003
8.400	.000001
8.600	.000001
8.800	.000000
9.000	.000000
9.200	.000000
9.400	.000000
9.600	.000000
9.800	.000000
10.000	.000000

Problem 4a.) Computes the solution to the Lorenz set of differential equations for case $\theta=60^\circ$. The driver program **hw3pr4a.f** makes use of numerical recipes, **rkdumb.f**, an extension of **rk4.f** that implements the fourth-order Runge-Kutta algorithm. The subroutine **derives6.f** was written for the Lorenz set with the perturbation terms f, θ , and g.

```

PROGRAM hw3pr4a
INTEGER nstep,nvar
PARAMETER (nvar=3,nstep=60000)
INTEGER i,j
REAL xi,yi,zi,x(60001),x1,x2,vstart(nvar)
EXTERNAL derivs6
common /path/ nhrs,xx(60001),y(3,60001)
xi=-6.
yi=-6.
zi=2.
x1=0.0
x2=300.
vstart(1)=xi
vstart(2)=yi
vstart(3)=zi
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs6)
OPEN(unit=45,file='hw3pr4.out',status='unknown')
c   write(45,*)'Number of function evaluations:'
c   write(45,*)'    time'','      x','      y','      z'
do 11 i=1,nstep
   write(45,'(1x,f10.4,2x,3f12.6)')xx(i),y(1,i),y(2,i),y(3,i)
11   continue
END

```

```

SUBROUTINE derivs6(x,y,dydx)
REAL x,y(*),dydx(*)
INTEGER nrhs
COMMON nrhs
nrhs=nrhs+1
dydx(1)=10.*(y(2)-y(1))+2.5*cos(60*3.14159/180.0)
dydx(2)=y(1)*(28.-y(3))-y(2)+2.5*sin(60*3.14159/180.0)
dydx(3)=y(1)*y(2)-8.*y(3)/3+2.
return
END

```

Problem 4a.) Computes the solution to the Lorenz set of differential equations for case $\theta=240^\circ$. The driver program **hw3pr4b.f** makes use of numerical recipes, **rkdumb.f**, an extension of **rk4.f** that implements the fourth-order Runge-Kutta algorithm. The subroutine **derives7.f** was written for the Lorenz set with the perturbation terms f, θ , and g.

```

PROGRAM hw3pr4b
INTEGER nstep,nvar
PARAMETER (nvar=3,nstep=60000)
INTEGER i,j
REAL xi,yi,zi,x(60001),x1,x2,vstart(nvar)
EXTERNAL derivs7
common /path/ nhrs,xx(60001),y(3,60001)
xi=-6.
yi=-6.
zi=2.
x1=0.0
x2=300.
vstart(1)=xi
vstart(2)=yi
vstart(3)=zi
CALL rkdumb(vstart,nvar,x1,x2,nstep,derivs7)
OPEN(unit=45,file='hw3pr4b.out',status='unknown')
c   write(45,*)'Number of function evaluations:'
c   write(45,*)'    time','      x','      y','      z'
do 11 i=1,nstep
   write(45,'(1x,f10.4,2x,3f12.6)')xx(i),y(1,i),y(2,i),y(3,i)
11  continue
END

```

```

SUBROUTINE derivs7(x,y,dydx)
REAL x,y(*),dydx(*)
INTEGER nrhs
COMMON nrhs
nrhs=nrhs+1
dydx(1)=10.*(y(2)-y(1))+2.5*cos(240*3.14159/180.0)
dydx(2)=y(1)*(28.-y(3))-y(2)+2.5*sin(240*3.14159/180.0)
dydx(3)=y(1)*y(2)-8.*y(3)/3+2.

```

```
return  
END
```

Problem 5a.) Flat plate temperature similarity solution (**Pr=0.714, air**). The driver program **hw3pr3.f** was modified for Pr.

OUTPUT: fn=hw3pr5a.out

Recovery Factor, $\theta(\eta=0) =$

.844722

eta	theta
.000	.844722
.200	.841573
.400	.832131
.600	.816438
.800	.794601
1.000	.766829
1.200	.733460
1.400	.694976
1.600	.652017
1.800	.605375
2.000	.555977
2.200	.504847
2.400	.453064
2.600	.401705
2.800	.351789
3.000	.304223
3.200	.259759
3.400	.218966
3.600	.182214
3.800	.149686
4.000	.121386
4.200	.097172
4.400	.076791
4.600	.059904
4.800	.046129
5.000	.035063
5.200	.026306
5.400	.019479
5.600	.014235
5.800	.010265
6.000	.007304
6.200	.005128
6.400	.003551
6.600	.002426
6.800	.001635
7.000	.001086
7.200	.000712
7.400	.000460
7.600	.000293
7.800	.000184
8.000	.000114
8.200	.000069

8.400	.000042
8.600	.000024
8.800	.000014
9.000	.000008
9.200	.000004
9.400	.000002
9.600	.000001
9.800	.000000
10.000	.000000

Problem 5.b.) Flat plate temperature similarity solution (**Pr=6.750, water**). The driver program **hw3pr3.f** was modified for Pr.

OUTPUT: fn=hw3pr5b.out

Recovery Factor, $\theta(\eta=0) =$

2.492125

<u>eta</u>	<u>theta</u>
.000	2.492125
.200	2.462354
.400	2.373335
.600	2.227093
.800	2.028960
1.000	1.788649
1.200	1.520416
1.400	1.241963
1.600	.972098
1.800	.727666
2.000	.520681
2.200	.356663
2.400	.234731
2.600	.149263
2.800	.092325
3.000	.055894
3.200	.033251
3.400	.019453
3.600	.011171
3.800	.006275
4.000	.003436
4.200	.001829
4.400	.000944
4.600	.000472
4.800	.000229
5.000	.000107
5.200	.000048
5.400	.000021
5.600	.000009
5.800	.000004
6.000	.000001
6.200	.000001
6.400	.000000
6.600	.000000
6.800	.000000

7.000	.000000
7.200	.000000
7.400	.000000
7.600	.000000
7.800	.000000
8.000	.000000
8.200	.000000
8.400	.000000
8.600	.000000
8.800	.000000
9.000	.000000
9.200	.000000
9.400	.000000
9.600	.000000
9.800	.000000
10.000	.000000

APPENDIX C - MATLAB PLOT CODE:

Problem 1a_1.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using RKDUMB.

```
%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:60000;
    X(n)=hw3pr1a1(n,2);
    Y(n)=hw3pr1a1(n,3);
end
plot(Y,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. y)');
xlabel('Y');
ylabel('x');
print -dps h3plaxy.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:60000;
    X(n)=hw3pr1a1(n,2);
    Z(n)=hw3pr1a1(n,4);
end
plot(Z,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. z)');
xlabel('z');
ylabel('x');
print -dps h3plaxz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:60000;
    Y(n)=hw3pr1a1(n,3);
    Z(n)=hw3pr1a1(n,4);
end
plot(Z,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. z)');
xlabel('z');
ylabel('y');
print -dps h3playz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:30000;
    T(n)=hw3pr1a1(n,1);
    X(n)=hw3pr1a1(n,2);
end
plot(T,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. time)');
xlabel('time');
ylabel('x');
print -dps h3plaxt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:30000;
    T(n)=hw3pr1a1(n,1);
    Y(n)=hw3pr1a1(n,3);
end
```

```

plot(T,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. time)');
xlabel('time');
ylabel('y');
print -dps h3playt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:30000;
    T(n)=hw3pr1a1(n,1);
    Z(n)=hw3pr1a1(n,4);
end
plot(T,Z,'-');
title('RKDUMB: Mod. Lorenz Set (z v. time)');
xlabel('time');
ylabel('z');
print -dps h3plazt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1a1.out
for n=1:1:10000;
    X(n)=hw3pr1a1(n,2)
    Y(n)=hw3pr1a1(n,3);
    Z(n)=hw3pr1a1(n,4);
end
plot3(X,Y,Z,'-');
title('RKDUMB: Mod. Lorenz Set Trajectory (3-D)');
xlabel('x');
ylabel('y');
zlabel('z');
grid on
print -dps h3pla3d.ps

```

Problem 1a_2.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using RKQS.

```

%Plot of modified Lorenz set trajectory using RKQS.
load hw3pr1a2.out
for n=1:1:3000;
    T(n)=hw3pr1a2(n,1);
    X(n)=hw3pr1a2(n,2);
end
plot(T,X,'-');
title('RKQS: Mod. Lorenz Set (x v. time)');
xlabel('time');
ylabel('x');
print -dps h3pla2xt.ps

%Plot of modified Lorenz set trajectory using RKQS.
load hw3pr1a2.out
for n=1:1:3000;
    T(n)=hw3pr1a2(n,1);
    Y(n)=hw3pr1a2(n,3);
end
plot(T,Y,'-');
title('RKQS: Mod. Lorenz Set (y v. time)');
xlabel('time');

```

```

ylabel('y');
print -dps h3pla2yt.ps

%Plot of modified Lorenz set trajectory using RKQS.
load hw3prla2.out
for n=1:1:3000;
    T(n)=hw3prla2(n,1);
    Z(n)=hw3prla2(n,4);
end
plot(T,Z,'-');
title('RKQS: Mod. Lorenz Set (z v. time)');
xlabel('time');
ylabel('z');
print -dps h3pla2zt.ps

%Plot of modified Lorenz set trajectory using RKQS.
load hw3prla2.out
for n=1:1:3000;
    X(n)=hw3prla2(n,2);
    Y(n)=hw3prla2(n,3);
end
plot(Y,X,'-');
title('RKQS: Mod. Lorenz Set (x v. y)');
xlabel('y');
ylabel('x');
print -dps h3pla2xy.ps

%Plot of modified Lorenz set trajectory using RKQS.
load hw3prla2.out
for n=1:1:3000;
    X(n)=hw3prla2(n,2);
    Z(n)=hw3prla2(n,4);
end
plot(Z,X,'-');
title('RKQS: Mod. Lorenz Set (x v. z)');
xlabel('z');
ylabel('x');
print -dps h3pla2xz.ps

%Plot of modified Lorenz set trajectory using RKQS.
load hw3prla2.out
for n=1:1:3000;
    Y(n)=hw3prla2(n,3);
    Z(n)=hw3prla2(n,4);
end
plot(Z,Y,'-');
title('RKQS: Mod. Lorenz Set (y v. z)');
xlabel('z');
ylabel('y');
print -dps h3pla2yz.ps

```

Problem 1a_3.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using BSSTEP.

```

%Plot of modified Lorenz set trajectory using BSSTEP.
load hw3prla3.out
for n=1:1:1000;

```

```

T(n)=hw3pr1a3(n,1);
X(n)=hw3pr1a3(n,2);
end
plot(T,X,'-');
title('BSSTEP: Mod. Lorenz Set (x v. time)');
xlabel('time');
ylabel('x');
print -dps h3pla3xt.ps

%Plot of modified Lorenz set trajectory using BSSTEP.
load hw3pr1a3.out
for n=1:1:1000;
    T(n)=hw3pr1a3(n,1);
    Y(n)=hw3pr1a3(n,3);
end
plot(T,Y,'-');
title('BSSTEP: Mod. Lorenz Set (y v. time)');
xlabel('time');
ylabel('y');
print -dps h3pla3yt.ps

%Plot of modified Lorenz set trajectory using BSSTEP.
load hw3pr1a3.out
for n=1:1:1000;
    T(n)=hw3pr1a3(n,1);
    Z(n)=hw3pr1a3(n,4);
end
plot(T,Z,'-');
title('BSSTEP: Mod. Lorenz Set (z v. time)');
xlabel('time');
ylabel('z');
print -dps h3pla3zt.ps

%Plot of modified Lorenz set trajectory using BSSTEP.
load hw3pr1a3.out
for n=1:1:1000;
    X(n)=hw3pr1a3(n,2);
    Y(n)=hw3pr1a3(n,3);
end
plot(Y,X,'-');
title('BSSTEP: Mod. Lorenz Set (x v. y)');
xlabel('y');
ylabel('x');
print -dps h3pla3xy.ps

%Plot of modified Lorenz set trajectory using BSSTEP.
load hw3pr1a3.out
for n=1:1:1000;
    X(n)=hw3pr1a3(n,2);
    Z(n)=hw3pr1a3(n,4);
end
plot(Z,X,'-');
title('BSSTEP: Mod. Lorenz Set (x v. z)');
xlabel('z');
ylabel('x');
print -dps h3pla3xz.ps

%Plot of modified Lorenz set trajectory using BSSTEP.
load hw3pr1a3.out

```

```

for n=1:1:1000;
    Y(n)=hw3pr1a3(n,3);
    Z(n)=hw3pr1a3(n,4);
end
plot(Z,Y,'-');
title('BSSTEP: Mod. Lorenz Set (y v. z)');
xlabel('z');
ylabel('y');
print -dps h3p1a3yz.ps

```

Problem 1b_1.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using RKDUMB with f and theta.

```

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    T(n)=hw3pr1b1(n,1);
    X(n)=hw3pr1b1(n,2);
end
plot(T,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. time),theta=60 deg.');
xlabel('time');
ylabel('x');
print -dps h3p1b1xt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    T(n)=hw3pr1b1(n,1);
    Y(n)=hw3pr1b1(n,3);
end
plot(T,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. time),theta=60 deg.');
xlabel('time');
ylabel('y');
print -dps h3p1b1yt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    T(n)=hw3pr1b1(n,1);
    Z(n)=hw3pr1b1(n,4);
end
plot(T,Z,'-');
title('RKDUMB: Mod. Lorenz Set (z v. time),theta=60 deg.');
xlabel('time');
ylabel('z');
print -dps h3p1b1zt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    X(n)=hw3pr1b1(n,2);
    Y(n)=hw3pr1b1(n,3);
end
plot(Y,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. y),theta=60 deg.');

```

```

xlabel('y');
ylabel('x');
print -dps h3p1b1xy.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    X(n)=hw3pr1b1(n,2);
    Z(n)=hw3pr1b1(n,4);
end
plot(Z,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. z),theta=60 deg.');
xlabel('z');
ylabel('x');
print -dps h3p1b1xz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    Y(n)=hw3pr1b1(n,3);
    Z(n)=hw3pr1b1(n,4);
end
plot(Z,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. z),theta=60 deg.');
xlabel('z');
ylabel('y');
print -dps h3p1b1yz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b1.out
for n=1:1:30000;
    X(n)=hw3pr1b1(n,2);
    Y(n)=hw3pr1b1(n,3);
    Z(n)=hw3pr1b1(n,4);
end
plot3(X,Y,Z,'-');
title('RKDUMB: Mod. Lorenz Set Trajectory (3-D), theta=60 deg.');
xlabel('x');
ylabel('y');
zlabel('z');
grid on
print -dps h3p1b13d.ps

```

Problem 1b_2.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using RKDUMB with f and theta.

```

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    T(n)=hw3pr1b2(n,1);
    X(n)=hw3pr1b2(n,2);
end
plot(T,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. time),theta=240 deg.');
xlabel('time');
ylabel('x');
print -dps h3p1b2xt.ps

```

```

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    T(n)=hw3pr1b2(n,1);
    Y(n)=hw3pr1b2(n,3);
end
plot(T,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. time),theta=240 deg.');
xlabel('time');
ylabel('y');
print -dps h3p1b2yt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    T(n)=hw3pr1b2(n,1);
    Z(n)=hw3pr1b2(n,4);
end
plot(T,Z,'-');
title('RKDUMB: Mod. Lorenz Set (z v. time),theta=240 deg.');
xlabel('time');
ylabel('z');
print -dps h3p1b2zt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    X(n)=hw3pr1b2(n,2);
    Y(n)=hw3pr1b2(n,3);
end
plot(Y,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. y),theta=240 deg.');
xlabel('y');
ylabel('x');
print -dps h3p1b2xy.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    X(n)=hw3pr1b2(n,2);
    Z(n)=hw3pr1b2(n,4);
end
plot(Z,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. z),theta=240 deg.');
xlabel('z');
ylabel('x');
print -dps h3p1b2xz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    Y(n)=hw3pr1b2(n,3);
    Z(n)=hw3pr1b2(n,4);
end
plot(Z,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. z),theta=240 deg.');
xlabel('z');
ylabel('y');

```

```

print -dps h3p1b2yz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr1b2.out
for n=1:1:30000;
    X(n)=hw3pr1b2(n,2);
    Y(n)=hw3pr1b2(n,3);
    Z(n)=hw3pr1b2(n,4);
end
plot3(X,Y,Z, '-');
title('RKDUMB: Mod. Lorenz Set Trajectory (3-D), theta=240 deg.');
xlabel('x');
ylabel('y');
zlabel('z');
grid on
print -dps h3p1b23d.ps

```

Problem 4a.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using RKDUMB with f, theta, and g=2.

```

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4.out
for n=1:1:30000;
    T(n)=hw3pr4(n,1);
    X(n)=hw3pr4(n,2);
end
plot(T,X, '-');
title('RKDUMB: Mod. Lorenz Set (x v. time),g=2.0,theta=60 deg.');
xlabel('time');
ylabel('x');
print -dps h3p4axt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4.out
for n=1:1:30000;
    T(n)=hw3pr4(n,1);
    Y(n)=hw3pr4(n,3);
end
plot(T,Y, '-');
title('RKDUMB: Mod. Lorenz Set (y v. time),g=2.0,theta=60 deg.');
xlabel('time');
ylabel('y');
print -dps h3p4ayt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4.out
for n=1:1:30000;
    T(n)=hw3pr4(n,1);
    Z(n)=hw3pr4(n,4);
end
plot(T,Z, '-');
title('RKDUMB: Mod. Lorenz Set (z v. time),g=2.0,theta=60 deg.');
xlabel('time');
ylabel('z');
print -dps h3p4azt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.

```

```

load hw3pr4.out
for n=1:1:30000;
    X(n)=hw3pr4(n,2);
    Y(n)=hw3pr4(n,3);
end
plot(Y,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. y),g=2.0,theta=60 deg.');
xlabel('Y');
ylabel('x');
print -dps h3p4axy.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4.out
for n=1:1:30000;
    X(n)=hw3pr4(n,2);
    Z(n)=hw3pr4(n,4);
end
plot(Z,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. z),g=2.0,theta=60 deg.');
xlabel('z');
ylabel('x');
print -dps h3p4axz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4.out
for n=1:1:30000;
    Y(n)=hw3pr4(n,3);
    Z(n)=hw3pr4(n,4);
end
plot(Z,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. z),g=2.0,theta=60 deg.');
xlabel('z');
ylabel('y');
print -dps h3p4ayz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4.out
for n=1:1:30000;
    X(n)=hw3pr4(n,2);
    Y(n)=hw3pr4(n,3);
    Z(n)=hw3pr4(n,4);
end
plot3(X,Y,Z,'-');
title('RKDUMB: Mod. Lorenz Set Trajectory (3-D), theta=60 deg., g=2.0');
xlabel('x');
ylabel('y');
zlabel('z');
grid on
print -dps h3p14a3d.ps

```

Problem 4b.) Plot code for solution to modified Lorenz set of equations (non-linear dynamics) using RKDUMB with f, theta, and g=2.

```

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;
    T(n)=hw3pr4b(n,1);

```

```

X(n)=hw3pr4b(n,2);
end
plot(T,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. time),g=2.0,theta=240 deg.');
xlabel('time');
ylabel('x');
print -dps h3p4bxt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;
    T(n)=hw3pr4b(n,1);
    Y(n)=hw3pr4b(n,3);
end
plot(T,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. time),g=2.0,theta=240 deg.');
xlabel('time');
ylabel('y');
print -dps h3p4byt.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;
    T(n)=hw3pr4b(n,1);
    Z(n)=hw3pr4b(n,4);
end
plot(T,Z,'-');
title('RKDUMB: Mod. Lorenz Set (z v. time),g=2.0,theta=240 deg.');
xlabel('time');
ylabel('z');
print -dps h3p4bz.t.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;
    X(n)=hw3pr4b(n,2);
    Y(n)=hw3pr4b(n,3);
end
plot(Y,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. y),g=2.0,theta=240 deg.');
xlabel('y');
ylabel('x');
print -dps h3p4bxy.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;
    X(n)=hw3pr4b(n,2);
    Z(n)=hw3pr4b(n,4);
end
plot(Z,X,'-');
title('RKDUMB: Mod. Lorenz Set (x v. z),g=2.0,theta=240 deg.');
xlabel('z');
ylabel('x');
print -dps h3p4bxz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;

```

```

Y(n)=hw3pr4b(n,3);
Z(n)=hw3pr4b(n,4);
end
plot(Z,Y,'-');
title('RKDUMB: Mod. Lorenz Set (y v. z),g=2.0,theta=240 deg.');
xlabel('z');
ylabel('y');
print -dps h3p4byz.ps

%Plot of modified Lorenz set trajectory using RKDUMB.
load hw3pr4b.out
for n=1:1:30000;
X(n)=hw3pr4b(n,2);
Y(n)=hw3pr4b(n,3);
Z(n)=hw3pr4b(n,4);
end
plot3(X,Y,Z,'-');
title('RKDUMB: Mod. Lorenz Set Trajectory (3-D), theta=240 deg.,
g=2.0');
xlabel('x');
ylabel('y');
zlabel('z');
grid on
print -dps h3p14b3d.ps

```